INTERNAL REPRESENTATIONS OF FORMULAS IN JAPANESE COMPUTER ALGEBRA SYSTEM GAL(Formula Manipulation and Its Applications to Mathematical Study)

AUTHOR(S):

Sasaki, Tateaki; Motoyoshi, Fumio

数理解析研究所講究録
551巻 1985年 180-191
100

INTERNAL REPRESENTATIONS OF FORMULAS IN JAPANESE COMPUTER ALGEBRA SYSTEM GAL

佐々木建昭 Tateaki Sasaki*) and 元吉文男 Fumio Motoyoshi**)

*) The Institute of Physical and Chemical Research (理研)
Wako-shi, Saitama 351, Japan

**) Electrotechnical Laboratory, Ministry of ITI (電総研)
Niihari-gun, Ibaraki 305, Japan

abstract

This paper describes internal representations of formulas in a computer algebra system GAL which we are developing. In particular, the emphasis is laid on the usefulness of expression tags, description of data structures of some common expressions, and explanation of a unique method of controlling arithmetic.

Key words and phrases: computer algebra system, internal representation.

§1. Introduction

In a previous paper [1], we described the objectives of GAL project, the basic system design, and the global structure of GAL. In this paper, we describe internal representations of many common expessions. These representations were well investigated and tested so far. The internal representations are related directly with many important properties of computer algebra system, such as performance, usefulness, and generality. Furthermore, once a system has been developed, it is very difficult to change the internal representations because so many system procedures are written crucially dependently on the internal representations. In addition to these points, we must also consider that GAL aims at attaining superficially conflicting two objectives, generality and high performance. We, therefore, designed the internal representations in GAL very carefully.

As for the internal representations in GAL, three points are noteworthy: The first is the definition of many kinds of canonical representations by introducing expression tags, the second is the introduction of simple structures into the prefix-form representations, and the third is a unique strategy of controlling arithmetic.

We find that expression tags are quite useful not only because they allow us to introduce many canonicals but also they make the system highly modular and extensible. In GAL, prefix-form representations are designed so that they represent formulas in quite a similar way as the human does. As a byproduct of this design as well as the strategy of controlling arithmetic, a high performance of arithmetic on prefix-forms is attained. In §6, we present some empirical data showing the performance of GAL.

§2. Canonicals and expression tags

The representations of formulas in GAL are classified into three classes, Basic-NUMbers, CANOnicals, and PREfix-Forms, where capital letters denote the abbreviated names. The BNF definition of the BNUM is

<BNUM> ::= <INTeger> | <RATional-NUMber> |

<FLOATing-number> | <INTerVaL>

We assume that these numbers are contained in the host Lisp system. (Currently, not every Lisp system contains RATNUM data type, but RATNUM will be available soon in most Lisp systems. The INTVL is available only in a few Lisp systems now, but the authors recommend INTVL as a useful data type in Lisp systems for scientific use.) The precision of the above numbers should be arbitrary.

By canonical representations in computer algebra systems, we mean some prescribed and unique data structures representing mathematical expressions. Most computer algebra systems define canonical representations for common mathematical expressions such as polynomials, although the representations are considerably different from system to system. There are several important merits of doing so. First, mathematically equivalent expressions are uniquely represented, making the equivalence check quite easy. Second, a high computation speed is attained. Third, fixing the data structure is very convenient for implementing many algebraic algorithms. The canonical data structures should also be defined for structured mathematical expressions such as matrix or set. In the GAL, canonical data structures are defined for ALGebraic-Numbers, POLYnomials, ALGebraic-Functions, RATional-Functions, SERIes, VECTors, MATrices, TENSors, SETs, and Ordered-SETs. (Currently, not all of these are implemented.) Furthermore, GRAPHs etc. are planned to be incorporated into GAL as canonicals.

In some systems, different canonicals are distinguished from each other by the difference of data structures. Since the GAL contains so many canonicals, this method will surely lead to confusion. In order to distinguish many canonicals clearly and efficiently, we put an expression tag at the head of each list representing a canonical. The expression tag is a Lisp short integer (normally 24 bits length), and it is composed of three parts, global type tag, local type tag, and domain-property tag. For the significance and usage of these tags, see [1]. In this paper, we want to emphasize the usefulness of expression tags for table-driven procedure call.

We explain this point further by an example. In the current GAL, the global ty tags of POLY and RATF are 2 and 4, respectively. The multiplication procedure f POLY and RATF is stored in TABLE*CANOCANO(2,4). Hence, given two CANOs multiplicands, we first check their global type tags and take out the necessa procedure from TABLE*CANOCANO. With this scheme of procedure call, we can program t procedures for one CANO almost independently of other CANOs. Hence, as for CANOs, G is highly modular and extensible. However, if we define canonical representations f most expressions in GAL, much memory will be expended for the tables such TABLE*CANOCANO.

§3. Data structures of common canonicals

In the following, by "bare-CANO" we mean a canonical data structure with t expression tag deleted. By "A . B" and "(E_1 ... E_n)" we mean the dotted pair of A a B and the list with elements E_1, ..., E_n, respectively. We use VAR and TRAN abbreviations of VARiable and TRAnscendental-Number, respectively.

The BNF definition of POLY data structure is as follows:

<bare-POLY> ::= (<main-VAR> <P-C pair> ... <P-C pair>) ;*)

<main-VAR> ::= <commutative VAR> | <TRAN> ;

<P-C pair> ::= <POWER> . <COEFF> ;

<POWER> ::= <nonnegative INT> ;

<COEFF> ::= <BNUM> | <ALGN> | <bare-POLY> ;

*) The main-VAR is the highest order VAR in the bare-POLY, and the <P-C pair> are in decreasing order.

For example, when the VAR x is of higher order than the VAR y, the canonic representation of $x^2 + 2xy + 1$ is

(#TAG x (2 . 1) (1 . (y (1 . 2))) (0 . 1)).

This data structure is, when the tag is deleted, quite similar to the polynomi representation in MACSYMA [2], although not exactly the same. It should be noted th

a TRAN is treated as a VAR in the internal representation. However, a TRAN is constructed to be of lower order than any VAR, hence the TRAN appears in coefficients of VARs. It should be noted further that functions such as sin(x) or cos(y) are not allowed as variables of polynomials in GAL.

Using POLY data structure, RATF data structure is defined as

<bare-RATF> ::= (<D-N pair> ... <D-N pair>) ;*)

<D-N pair> ::= <DENOM> . <NUMER> ;

<DENOM> ::= <bare-POLY> | <BNUM> ;

<NUMER> ::= <BNUM> | <ALGN> |
<bare-POLY> | <ALGF> ;

*) The <D-N pair>s are in increasing order.

The pair of each denominator and numerator is determined by preserving the input form, unless the user specifies the collection of fractions. However, the common monomials in the numerator and denominator are cancelled, and the fractions of the same denominator are collected into a single fraction. Therefore, the RATF data structure is not canonical in a strict sense, because there are many different data structures which are mathematically equivalent. For example, $1/(xy+2x+y+2) - 1/(x+1) - 1/(y+2)$, which is equal to 0, is represented as

(#TAG (y+2 . -1) (x+1 . -1) (xy+2x+y+2 . 1)),

where underlines denote bare-POLY data structures.

The POLY data structure is used also in ALGN and ALGF:

<bare-ALGN> ::= <bare-ALGF> ::= <NUMER> . <DENOM> ;

<NUMER> ::= <BNUM> | <bare-POLY> ;*)

<DENOM> ::= <BNUM> | <bare-POLY> ;*)

*) The VARs in <bare-POLY> are now algebraic identifiers.

For ALGN and ALGF, we can always normalize the DENOM to 1. However, the normalization is often time consuming and may lead to complicated forms. Hence, we allow unnormalized representations. In this sense, the representations of ALGN and ALGF are

not truely canonical.

In determining the data structures of POLY and RATF, we considered the hig performance to be the most important condition. In fact, if the RATF is represente by a single fraction, which is canonical when the common factor in the denominator an numerator is cancelled, the representation will become large in general. Hence, ou representation of RATF will lead to a much higher performance than that in REDUCE [3] for example. On the other hand, elegance of mathematical treatment and the generalit of representations are the leading conditions in determining the data structures c ALGN and ALGF. For example, the imaginary number i is represented in the followin complicated form (#I is the GAL name of i)

(#TAG (#I (1 . 1)) . 1).

However, representing i in this way, we can treat the i just the same as othe algebraic numbers. (If a user wants to manipulate i efficiently, he can introduce variable, say I, with the simplification rule I**2 -> -1.)

§4. Data structures of prefix-forms

Although the canonical representations are quite useful, we must introduce als prefix-forms into GAL. There are mainly two reasons for doing so. The first is tha canonical representations are not able to represent various forms of expressions whic are often necessary in actual calculations. For example, we often need a factore form $(x + 1)^2(x + 2)^2$ or $(x^2 + 3x + 2)^2$ instead of $x^4 + 6x^3 + 13x^2 + 12x + 4$. Th second reason is that the existence of canonical form is unknown for many mathematica expressions.

The first element of the list representing the PREF is an identifier. Hence, th PREF can be distinguished clearly and easily from the CANOs the first elements o which are integers. All the prefix-forms in GAL are classified as follows:

<PREF> ::= <RATional-EXPRession> | <SET-EXPRession> |
<RELational-EXPRession> | <LOGical-EXPRession> .

The first elements of the lists representing SETEXPR, RELEXPR, and LOGEXPR are, respectively, set theoretic operators $\cup$, $\cap$, etc., relational operators =, >, etc., and logical operators V, $\wedge$, etc. The RATEXPR is constructed in the following way:

(1) It may contain BNUMs as numbers;

(2) It may contain VARs (commutative or noncommutative) and TRANs;

(3) It may contain unstructured CANOs (hence, ALGN, POLY, ALGF, RATF, and SERI);

(4) It may contain mathematical FUNCtions (commutative or noncommutative);

(5) It may contain OPERators;

(6) Operation of addition, multiplication, division, exponentiation, and composition (of functions as well as operations) are allowed in any nested way.

Although the RATEXPRs cover very wide classes of expressions, they are classified into only three forms, C*-form, NC*-form, and C+-form. The C*-form and NC*-form represent products of factors which are mutually commutative and noncommutative, respectively. The C+-form represents the sum of BNUM, CANOs, NC*-forms, C*-forms, and C+-forms. We give the BNF definitions of these forms.

<C+-form> ::= (C+ <BNUM> <CANO> ... <CANO>
<T-C pair> ... <T-C pair>) ;*1)

<T-C pair> ::= <prefix-TERM> . <numeric-COEF> ;

<prefix-TERM> ::= <NC*-form> | <C*-form> |
<commutative C+-form> ;*2)

<numeric-COEF> ::= <BNUM> ;

<C*-form> ::= (C* <CBASE> <POWER> ... <CBASE> <POWER>) ;*3)

<NC*-form> ::= (NC* <NCBASE> <POWER> ... <NCBASE> <POWER>) ;

<CBASE> ::= <BNUM> | <unstructured CANO> |
<commutative FUNC> | <NC*-form> |
<commutative C+-form> ;*4)

<NCBASE> ::= <noncommutative VAR> | <OPER> |
<noncommutative FUNC> |

<NC*-form> | <noncommutative C+-form> ;

<POWER> ::= <BNUM> | <unstructured CANO> | <RATEXPR> ;

<FUNC> ::= <FUNC-name> . (<ARG> ... <ARG>) ;

<ARG> ::= nil | <BNUM> | <CANO> | <RATEXPR> ;

*1) The <BNUM> and <CANO>s may be nil, <CANO>s are in increasing order, and <T-C pair>s are in decreasing order;

2) Ordering of constructors is NC > C* > C+ ;

*3) The <CBASE>s are in decreasing order;

4) The number of <NC-form> is 1 at most.

In most systems, "+", "-", "*", "/", and "**" are used as constructors of prefix-forms. In our representations, three of these are eliminated by introducing simple structures into prefix-forms. This structure introduction allows us to arrange the terms in C+-forms or factors in C*-forms in quite a natural order. Furthermore, arithmetic on RATEXPRs becomes quite simple. Note that the C+-form can contain C*-forms and NC*-forms simultaneously, which is necessary because we often handle sums of noncommutative and commutative terms. Since the constructor NC* is of higher order than C* and C+, and since the C+-form does not contain noncommutative C+-form as a term, NC*-forms are arranged near the head of the C+-form. Hence, we can easily check whether or not a given C+-form contains noncommutative elements.

The followings are some examples of RATEXPR representations, where underlines denote CANO representations, "sin" is assumed to be of high order than "cos", and "A" and "B" are noncommutative VARs:

$\sin(x)^2\cos(x)$ ==> (C* (sin $\underline{x}$) 2 (cos $\underline{x}$) 1) ;

$(2x^2+1)(x+1)^2$ ==> (C* $\underline{(2x^2+1)}$ 1 $\underline{(x+1)}$ 2) ;

$(x+1)^3$A·B ==> (C* (NC* A 1 B 1) 1 $\underline{(x+1)}$ 3) ;

$4\sin(x)^3 + 3\cos(x)^4/(x+2) + (x^2+x+1)$ ==>

(C+ $\underline{(x^2+x+1)}$ ((C* (sin $\underline{x}$) 3) . 4)((C* (cos $\underline{x}$) 4 $\underline{(x+2)}$ -1) . 3)) .

It should be commented that using canonical representations for the arguments of FUNC

is not a waste of memory, because the polynomial data structure for a single variable is made unique as far as possible.

§5. Arithmetic control

For controlling arithmetic, such as whether or not a factored form is expanded, GAL provides a useful and practical method. The principles for arithmetic control in GAL are structure preserving and simulating human's method. In fact, the representation of RATF has been determined by the principle of structure preserving. These principles lead us to the following unique method of controlling arithmetic.

For the addition operation, GAL allows three modes, UNCOLlection, COLlection, and STrong-COLlection. Let u and v be nonzero BNUM, CANO, or RATEXPR. In UNCOL mode, addition of u and v results normally in a C+-form unless both u and v are BNUMs. For example, if u and v are CANOs such that u is of higher order than v, we have

u + v ==> (C+ v u).

In the COL mode (default), any BNUMs and CANOs are unified, and any C+-forms are unified to a single C+-form unless the result reduces to a simpler form, where CANOs in C+-forms are unified only when they are of the same type. Important is the addition of C*-forms. Two C*-forms are unified when they differ from each other by a single CANO/BNUM factor of power 1. For example,

$\sin(x)^3(x+1)(x-1) + 2\sin(x)^3(x-1)(y+2)$ ==> $\sin(x)^3(x+2y+5)(x-1)$,

but $\sin(x)^3(x+1)^2(x-1)$ and $\sin(x)^3(x-1)^2(y+2)$ are not unified. On the other hand, in STCOL mode, any BNUMs and CANOs in C+-forms are unified, and any C*-forms are unified when they differ from each other by a single CANO/BNUM factor of integer power. For example,

$\sin(x)^3(x+1)^2(x-1)^2 + \sin(x)^3(x-1)^2(y+2)$

==> $\sin(x)^3(x-1)^2(x^2+2x+y+3)$.

For the multiplication and exponentiation operations, GAL allows also three modes, UNEXPansion, EXPansion, and STrong-EXPansion. Let u and v be nonzero BNUM, CANO, or

RATEXPR. In UNEXP mode, exponentiation of u results normally in a C*-form and multiplication of u and v does also unless both u and v are BNUMs. For example, if u and v are CANOs such that u is or higher order than v, we have

$$u \times v \quad ==> \quad (C* \ u \ 1 \ v \ 1).$$

In the EXP mode (default), CANOs and BNUMs are multiplied as usual and C+-forms are expanded, but any CANOs and C+-forms in C*-forms are not expanded. For example,

$$\sin(x)^2(x+1)^2(z+3) \times \cos(y)^3(x+1)(y+2)$$
$$==> \quad \sin(x)^2\cos(y)^3(x+1)^3(y+2)(z+3).$$

On the other hand, in STEXP mode, all the BNUMs, CANOs, and C+-forms of integer powers are expanded.

Thus, once factored forms or power forms are constructed, their structures are preserved until the user issues COLLECT/EXPAND command or adds/multiplies them in STCOL or STEXP modes. (The COLLECT command and STCOL mode do not always destroy the structure.) Note that, the arithmetic in COL and EXP modes is almost the same as that in our paper-and-pencil calculations. One practical problem in calculating large expressions by computer is how to avoid the expression swell, and a human solves this problem by structure preserving. Our method of arithmetic control seems to be a simple and useful, although not sufficient, solution to this problem.

§6. Performance

We compared GAL with REDUCE-2 and REDUCE-3 in the speed of basic arithmetic. The following timing data were obtained on a FACOM/M-380 computer, where CLISP system [4] is used for GAL and REDUCE-2 and SLISP system [5] for REDUCE-3. In the following table, expressions S, A, B, C, D, E, and F are

$$S = 2x^{37} + 3x^{33} + 4x^{27} + 5x^{21} - 6x^{13} - 7x^{9} - 8x^{5} - 9x^{3} - 10,$$

$$A = x^2y + xy^2 + y^2z^2 + z + 1, \quad B = A + yz,$$

$$C = (x + y + u)(x + z - v) + x + u, \quad D = C + y - v,$$

$$E = \sin(x)(x+y) + \cos(y)(z+1) + (x-y), \quad F = E + (z+1).$$

unit/msec*)	GAL	REDUCE-2	REDUCE-3
S**7	211	3,582	540
A7 = A**7	65	182	165
B7 = B**7	126	340	315
A7 * B7	4,292	21,791	18,409*2)
A7 / A**5	61*1)	****	211*2)
B7 / B**2	151*1)	****	506*2)
C4 = C**4	103	166	190
D4 = D**4	162	219	261
C4 * D4	9,045	19,194	25,889
E6 = E**6	103*3)	126	150
F6 = F**6	278*3)	336	398
E6 * F6	10,038*3)	20,995	*****5)
E6 * F6	3,487*4)	same	****

**) the result is not printed out,
*1) using DIVIDE command,
*2) using REMAINDER command,
*3) in STEXP mode,
*4) in EXP mode,
*5) GBC-overflow.

In GAL, polynomials A, B, C, D are represented in canonical form and expressions E and F are represented in prefix-forms. On the other hand, in REDUCE, E and F are represented in canonical forms by treating sin(x) and cos(y) as variables. We see that performance of GAL is pretty good. It is remarkable that arithmetic on prefix-forms is quite efficient in GAL. Usually, performance of arithmetic on prefix-forms is considerably worse than that on canonicals. We see furthermore that arithmetic control in GAL is able to make the calculation on prefix-forms quite efficient.

Acknowledgement. We thank Prof. S. Watanabe, Dr. Y. Kanada, and Mr. H. Murao for discussions on the representations of polynomials.

References

[1] T. Sasaki and A. Furukawa, "Design of a general computer algebra system," IPSJ Meeting Report on Symbol Manipulation, No.23-2, March, 1983.

[2] A. C. Hearn, "REDUCE user's manual", Version 3.0, The Rand Corporation, 1983.

[3] The MATHLAB Group, "MACSYMA Reference manual", 9th Version, Laboratory for Computer Science, MIT, 1977.

[4] A. C. Norman, "Introduction to the Cambridge Lisp System", Computer Laboratory, Univ. Cambridge, 1981.